Evaluating the FITTEST Automated Testing Tools in SOFTEAM: an Industrial Case Study

Etienne Brosse

Alessandra Bagnato

Tanja E. J. Vos

Nelly Condori-Fernandez

Technical Report UU-CS-2014-009
May 2014

Department of Information and Computing Sciences
Utrecht University, Utrecht, The Netherlands
www.cs.uu.nl

ISSN: 0924-3275

Department of Information and Computing Sciences
Utrecht University
P.O. Box 80.089
3508 TB Utrecht
The Netherlands

Evaluating the FITTEST Automated Testing Tools in SOFTEAM: an Industrial Case Study

Etienne Brosse, Alessandra Bagnato
SOFTEAM
Paris, France
Email: {etienne.brosse, alessandra.bagnato}@softeam.fr

Tanja E. J. Vos, Nelly Condori
Universidad Politécnica de Valencia
Valencia, Spain
Email: {tvos, nelly}@pros.upv.es

***Abstract*—[Context] Many test automation tools are currently available. However, most of them do not yet support automated test case design and evaluation. [Method] The FITTEST EU project has developed such automated tools. This paper aims at their evaluation within an industrial case study. The case study was conducted at SOFTEAM for testing Modelio SaaS, a web administration console written in PHP, which allows an administrator to connect to his account for managing modeling projects created with SOFTEAMs Modelio UML Modeling tool. [Objective] This case study has investigated whether current SOFTEAM testing practices could be improved or complemented by using some of the automated testing tools that were developed within the FITTEST EU project. [Results] Although the existing Test Suite from SOFTEAM (TS_{soft}) that was selected for comparison is substantially smaller than the Test Suite generated by FITTEST ($TS_{fittest}$), the effectiveness of $TS_{fittest}$, measured by the injected faults coverage is significantly higher (50% vs 70%). With respect to efficiency, .. [Conclusions] Within SOFTEAM and for the testing of the target product in SOFTEAMs testing environment: the FITTEST tools can increase the effectiveness of the current practice and the test cases automatically generated by the FITTEST tools can help in more efficient identification of the source of the identified faults. Moreover, the FITTEST tools have shown the ability to automate testing within a real industry case.**

I. Introduction

Software testing is the process of executing a program or system with the intent of finding defects [9]. It is currently the most important and widely used quality assurance technique applied in the industry. It may require over 50% of development budget and time [1]. Many test automation tools are currently available to aid test planning and control as well as test case execution and monitoring [4]. However, most of these tools, particularly those used in industrial practice, share a similar passive philosophy towards test case design, selection of concrete test data and test evaluation (i.e. oracles). They leave these crucial, time-consuming and demanding activities to the human testers. The lack of automation of these important testing activities means that the industry spends much effort and money on testing, nevertheless the quality of the resulting tests is sometimes low since they fail to find important errors in the system.

FITTEST, and EU funded research project, aims to overcome, at least to some extent, this problem. The FITTEST project involves a consortium of diverse competence, from execution monitoring, model-based testing [3], combinatorial testing [11], to search-based software engineering [6]. The ultimate goal of the project is to develop an Integrated Testing Environment (ITE for brevity) that consists of a suite of pluggable components that are being integrated for the FITTEST automated and continuous testing approach and tool set[14]. Being continuous, this testing approach will be suitable for testing fast evolving applications or those with dynamic and adaptive behaviors. i.e. the types of systems are envisaged to run on the Future Internet.

In this paper we present a case study for evaluating a subset of the FITTEST components, specifically those components that are responsible for Automated Test Case Design and Evaluation. The case study has been executed at the company is called SOFTEAM , a private software vendor and engineering company with about 700 employees located in Paris, France and a partner of the FITTEST project. Experimental evaluation of some of the techniques implemented in the FITTEST techniques have already been conducted and presented in earlier work [10]. Moreover a case study executed at IBM Research has been published []. This study aims to obtain more empirical evidence of the applicability of these techniques within the context of an industry team testing an industrial system. To this end we present a "which is better" type of case study [8]. These case studies are powerful since, although they cannot achieve the scientific rigor of formal experiments, the results of a case study can provide useful insights to help others judge whether the specific technology being evaluated could benefit their own organization. In order to assess tools, evaluative case study research must involve realistic systems and realistic subjects, as explicitly done in this study.

The contribution of this paper is twofold. On the one hand, it describes more promising results of the use of automated test tool on an industrial case study. On the other hand, the study in this paper can serve as an example that others can follow when encountering the need to evaluate an automated testing tool.

The remainder of the paper is organized as follows: Section II describes the industrial context in which the study was performed. Section III presents the case study design framework, Finally, Section IV presents the results and Section V concludes the paper.

II. Context- where was the case study performed

This study has been executed at SOFTEAM. One of the priorities of SOFTEAM is to maximize feature coverage of

their test suites with minimum costs. Specifically those test suites created for the Modelio SaaS system.

Modelio SaaS is a web administration console written in PHP, which allows an administrator to connect to his account for managing modeling projects created with Modelio UML Modeling tool [1]. Modelio SaaS, besides the customer users, provides services to different roles of administrators: server administrators, account managers and project managers.

III. Design of the case study

A. Objective - What to achieve?

As indicated before, SOFTEAM wanted to evaluate the FITTEST automated testing tools to see if they are applicable to a selected System Under Test and how they compare to current practice. What makes a testing tool applicable in industry? First of all, it should be effective in finding faults! Second, this should be done efficiently, i.e. in a reasonable amount time. Finally, although finding faults in a reasonable amount of time is important, the amount of effort to set up and use the testing tools in the testing processes currently implanted should be important too. Hence, following [13], we focus on:

- ***RQ1*** [**Effectiveness and Efficiency**] Compared to the current test suite used for testing at SOFTEAM, can the FITTEST technologies contribute to the effectiveness and efficiency of testing when it is used in the testing environments at SOFTEAM?
- ***RQ2*** [**Effort**] How much effort would be required to deploy the FITTEST technologies within the testing processes implanted at SOFTEAM?
- ***RQ3*** [**Subjective Satisfaction**] How satisfied are SOFTEAM testing practitioners during the learning, installing, configuring and usage of the technique when it is used in their real testing environments?

B. Cases or Treatments - What are being studied?

1) Current test case design techniques used at SOFTEAM: Modelio SaaS' testing and development team consists of 1 product director, 2 developers and 3 research engineers who all participate in the testing process. The testing practice at Softeam is to create test cases by relying on specified use cases. Each test case describes a sequence of the user interactions with the graphical user interface as shown by Figure 2.

The test cases are managed with the *TestLink*[1] software and grouped as test suites according to the part of the system that they enable to test. All them are executed manually by a test engineer. If a failure occurs, the test engineer reports it to the *Mantis*[2] bug tracking system and assigns it to the developer in charge of the part affected by the failure. He also provides the Apache log file for the web UI as well as the Axis log file for the web services. Then, Mantis mails the developer in charge of examining/fixing the reported failure.

It took approximately 7 work days to design and build the test suite and together, the testers need an hour to execute all of the 51 test cases if no errors occur.

[1]http://sourceforge.net/projects/testlink/

[2]http://www.mantisbt.org/

Softeam's testing process in projects other than Modelio SaaS is similar. A tester has access to the project specifications (most of the time a textual description). From this specification, he manually creates a first test suite under *TestLink* and then executes it. According to its results he will modify the test suite, enter failure(s) into *Mantis*, discuss the issues with the development team and re-execute the modified test cases, etc.

2) The FITTEST tools for automated test case design and evaluation: The FITTEST testing approach is shown in Figure 1 and contains four phases:

1) *Logging* - Run the target application (SUT) and collect the logs it generates. This can be either real usage by end users of the application in the production environment, or test case execution in the test environment.
2) *Test-ware generation* - Analyse the logs to infer different testwares (i.e. FSM models, Oracles, Domain Input Specifications (DIS) and Test Cases).
3) *Test Execution* - The test cases are executed by running the SUT.
4) *Test evaluation* - The outcome of the running the test cases is evaluated using the oracles that are available.

For this case study, we instantiated two key components and their underpinning techniques of FITTEST. The two components are those responsible for Automated Test Case Design and Evaluation: **Logs2FSM**, **FSM2CT** and **CT2Selenium**, that are done during the second phase (i.e. *Test-ware generation*):

- **Logs2FSM**, this component takes the logs generated by running Modelio SaaS and infers FSM models by applying the k-tail event-based model inference approach [2]. The *model-based oracles* that also result from this tool (see Figure 1) refer to the use of the paths generated from the inferred FSM as oracles. If these paths, when transformed to test cases, cannot be fully executed, then the tester needs to inspect the failing paths to see if that is due to some faults, or the paths themselves are infeasible.
- **FSM2CT**, this component takes the output models of **Logs2FSM** and a Domain Input Specification (DIS) file created by a tester for Modelio SaaS to generate concrete test cases. This component implements the *M*C* technique that combines model-based testing and combinatorial testing presented in [10]. This basically consist in: (1) generating test paths from the FSM (using algorithms that range from simple graph visit algorithms, to advanced techniques based on maximum diversity of the event frequencies, and semantic interactions between successive events in the sequence); (2) transform these paths into classification trees using the Classification Tree Editor (CTE XL)[3] [5] format, enriched with Domain Input Specifications (DIS) such as data types and partitions; (3) generate test combinations from those trees using t-way combinatorial criteria.
- **CT2Selenium**, this component take classification trees and domain input specification file as input and gener-

[3]http://www.berner-mattner.com

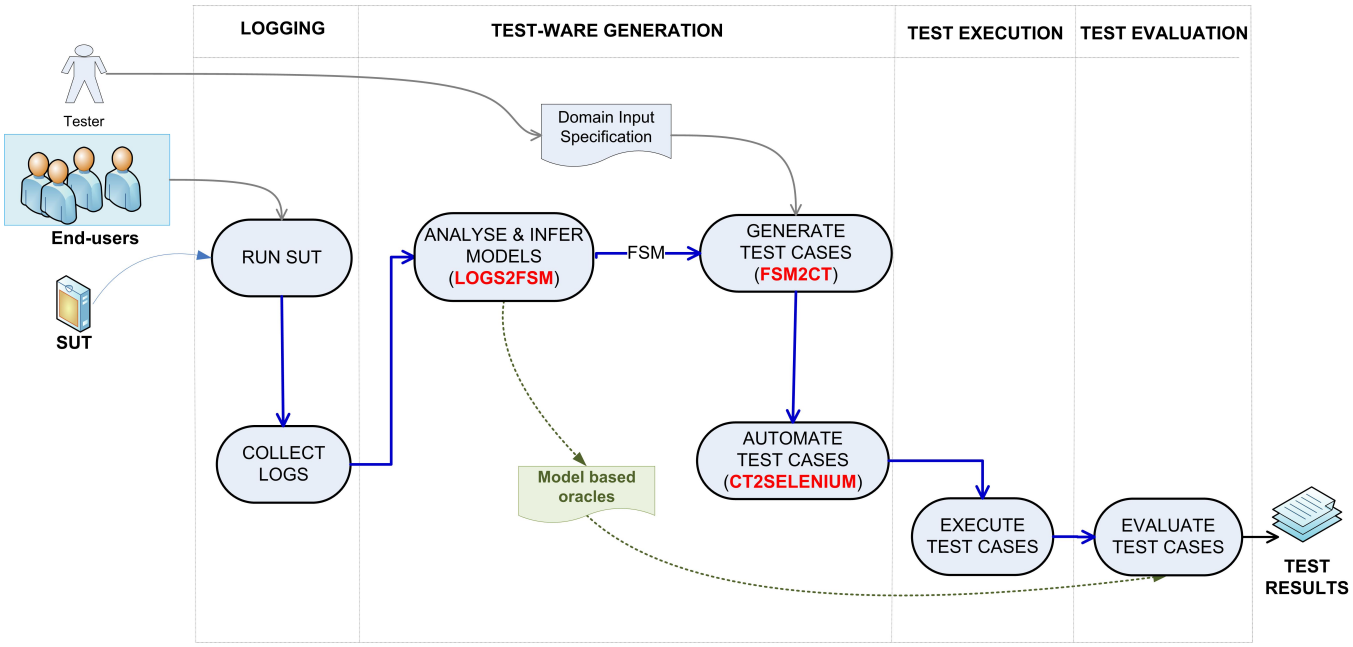

Fig. 1. The control flow of the FITTEST Automated Test Case Design and Evaluation. It contains 4 phases: *Logging*, *Test-ware generation*, *Test execution*, and *Test evaluation*.

ate executable test cases in the format of Selenium/JUnit.

C. Objects of the study: the SUT and the injected faults

1) The System Under Test (SUT): The SUT selected for this study is the Modelio SaaS system[4], which is a prototype developed at Softeam. Modelio SaaS is a PHP web application, that allows for easy and transparent configuration of distributed environments. It can run in virtual environments on different cloud platforms, offers a large number of configuration options and hence poses various challenges to testing. The source code is composed of 50 PHP files with a total of 2141 lines of executable code.

In the case study we will focus on the web administration console, which allows administrators to manage various aspects of projects created with the Modelio modeling tool. In rpinciple there are 3 different roles:

- A Server administrator: The main task of a server administrator is to create Modelio Workgroup Server and assign a Account Manager for each of them.
- B Account Manager: its main roles are to administrate Modelio Workgroup Server including the creation Modelio Project Server. This kind of user is able to assign Project Manager to given Modelio Project Server and define the different artifact (module and component) available on each of its Modelio Workgroup Server.
- C Project Manager: administrate Modelio Project Server i.e. deploy (or not) artifact available on a Modelio Workgroup Server to specific the projects that he manage, assign role to people, etc.

[4]http://www.modelio.org/

2) TS_{Soft} – Softeam's existing manual Test Suite: The existing test suite is a set of 51 manually crafted system test cases that Softeam uses to perform regression testing of new releases. Each test case describes a sequence of user interactions with the graphical user interface as well as the expected results. Figure 2 shows an example of such a test case.

3) Injected Faults: For the sake of comparing fault finding capability, the testers of Modelio SaaS, have identified and selected 17 faults divided into 3 categories that resemble realistic faults that could normally be encountered when testing the system. These faults have been injeted into the system in order to be able to detrmine the fault finding capability of the existing test suite from SOFTEAM and the test suite generated by FITTEST.

D. Subjects - Who apply the techniques?

There were 4 subjects involved in this study. (1) Dr Marcos Almeida is a R&D engineer with 2 years of experience in modelling. He is mainly involved in Model Driven Engineering, Software and Business Processes Modelling. (2) Mr Antonin Abherve is a senior research engineer with more than 8 years of experience in a large panel of modelling domain e.g. Model Driven Engineering, Model Driven Architecture, Software Modelling, Service Oriented Architecture, Enterprise Architecture, Business Processes, etc. (3) Mr Etienne Brosse is a senior research engineer with more than 7 years of experience mainly in System Architecture Modelling (i.e. SysML, MARTE modelling), Model Driven Engineering, and Model Driven Architecture. (4) Dr Alessandra Bagnato is a research scientist within the Softeam R&D Department. She has been working as Project Manager related to software/service, embedded system, security and testing modelling in several European research project.

Test Case SaaS-7: Create a customer account from the administrator account		
Author:	cba	
#:	Step actions:	Expected Results:
1	Sign in as a server administrator	
2	Go to "Compte clients" -> "Créer un compte client"	The form to fill customer's details is displayed.
3	Fill: 1. the 'nom' field with a name 2. the "date de soucription" field with a date with format 'YYYY-MM-DD" 3. the "date de validité" field with a date with format 'YYYY-MM-DD" 4. the 'login' field with a login 5. the 'mot de passe' field with a password 6. the 'e-mail' field with an email address	
4	Click on 'Créer compte'	The 'Gestion des comptes clients" page must be displayed with a table containing the customer's details.
Execution type:	Manual	
Keywords:	None	

Fig. 2. Manual test case, used by Modelio SaaS testers for functional testing.

ID	Part of the Application	FileLocation	Description	Severity
1	Controller	AccountController.php line 102	When clicking on ”no” for account deletion confirmation, the system nevertheless deletes the account	Major
2	Controller	LoginController.php line 8	No login fields on login page	Block
3	Controller	ProjectsController.php line 8	Empty page when accessing the project creation page	Block
4	Controller	RamController.php line 31	Description is not added to the database when creating a component	Minor
5	Controller	RolesController.php line 48	Page not found error after editing a role	Major
6	Model	DeploymentInstance.php line 10	An error occurred” message when trying to view properties of a project	Block
7	Model	Module.php line 21	”An error occurred” message when trying to add a module to a project	Block
8	Model	Module.php line 34	”An error occurred” message when trying to upload a new module	Block
9	Model	Project.php line 36	”An error occurred” message when trying to view managed project	Block
10	Model	ProjectModule.php line 29	”An error occurred” message when trying to view properties of a project	Block
11	View	ComponentSelection line 19	Empty page when trying to add a component to a project	Block
12	View	Modules.php line 18	Allow empty content for module	Minor
13	View	ModuleSelection.php line 30	Empty page when trying to add a module to a project	Block
14	View	RoleSelection.php lines 27 to 30	”An error occurred” when trying to edit the role of a user of a project	Block
15	View	Server.php line 42	The type of the server is missing	Major
16	View	ServerSelection.php line 13	Empty form when trying to move a server	Block
17	View	Users.php line 82	Editing is possible when accessing through view link and vice versa	Minor

TABLE I. INJECTED FAULTS

E. Protocol

We adopted the following steps in this case study. See Figure 3. We disthinguished three different roles A (the server administrator role), B (the acound administrator role) and C (the project managers role). And four users: 1 user (role A + B) and 3 users for role C.

1) Install and configure the ITE
2) Create the three sets of logs: Logs_A, Logs_B and Logs_C.
3) Select test suite TS_{soft}
4) Generate three Test Suites with the FITTEST tools $\text{TS}^A_{fittest}$, $\text{TS}^B_{fittest}$ and $\text{TS}^C_{fittest}$
   a) Generate the three FSMs with Logs2FSM (FSM_A, FSM_B and FSM_C).
   b) Define three Domain Input Specification (DIS_A, DIS_B, DIS_C)
   c) Generate the concrete test data with FSM2Tests and create the three test suites.
5) Select and inject the faults
6) Execute TS_{soft} and measure
7) Execute $\text{TS}^A_{fittest}$, $\text{TS}^B_{fittest}$ and $\text{TS}^C_{fittest}$ and measure

F. Variables - What are being measured?

The *independent variables* of the study setting are: The FITTEST testing techniques; the complexity of the industrial system; TS_{soft}; the level of experience of testers of SOFTEAM that will use the techniques; the injected faults. The *dependent variables* are related those for measuring the applicability of the FITTEST tools in terms of effectiveness, efficiency, effort and subjective satisfaction. Next we present their respective defined metrics:

1) Measuring effectiveness:
   a) amount of injected faults detected by both TS_{soft} and , $\text{TS}^A_{fittest}$, $\text{TS}^B_{fittest}$ and $\text{TS}^C_{fittest}$.
   b) type of faults detected by both TS_{soft} and $\text{TS}^A_{fittest}$, $\text{TS}^B_{fittest}$ and $\text{TS}^C_{fittest}$.
   c) code covered by both TS_{soft} and $\text{TS}^A_{fittest}$, $\text{TS}^B_{fittest}$ and $\text{TS}^C_{fittest}$.
2) Measuring efficiency. For both TS_{soft} and $\text{TS}^A_{fittest}$, $\text{TS}^B_{fittest}$ and $\text{TS}^C_{fittest}$:

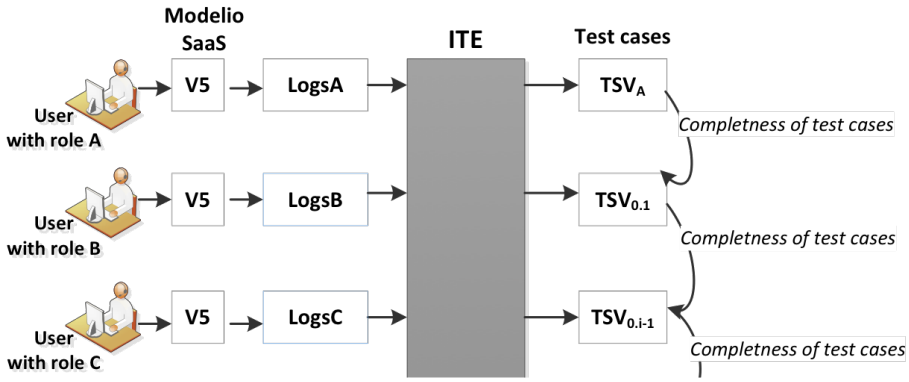

Fig. 3. We disthinguished three different roles A, B and C.

a) size of the test suites: number of test cases and number of events (or commands)
b) time needed to execute both the test suites

Moreover for $TS^A_{fittest}$, $TS^B_{fittest}$ and $TS^C_{fittest}$. we will measure:

c) number of traces and acerage trace length in $Logs_A$, $Logs_B$ and $Logs_C$ that were used to infer teh FSMs
d) metrics about the created FSMs (i.e. FSM_A, FSM_B and FSM_C): number of states, number of different events, number of transitions

3) Measuring effort in time (hours) for each of the subjects that was needed to create TS_{soft} and by both TS_{soft} and $TS^A_{fittest}$, $TS^B_{fittest}$ and $TS^C_{fittest}$.
4) Measuring subjective satisfaction through interviews and working diaries.

IV. Results and Discussion

A. Results

This section summarizes and discusses the results and outcomes of this case study. We have followed the protocol presented in the previous section to measure the variables identified for the involved test suites: TS_{soft}, $TS^A_{fittest}$, $TS^B_{fittest}$ and $TS^C_{fittest}$. Table II contains the measures specific for the generated FSM models that were used to generate the FITTEST test suites $TS^A_{fittest}$, $TS^B_{fittest}$ and $TS^C_{fittest}$. For illustrative reasons, part of this FSM can be found in Figure 4 and an example of a generated test sequence and corresponding test cases from the CTE XL[5] can be found in Figure 5. The figures give an idea how the models and test cases are visualized. They are generated and used automatically without human intervention. However, the tester can use graphical tools like CTE XL to enhance the models and tests, e.g. to put dependency on input data, if needed.

In Table V the descriptive measures for all test suites are listed.

Figure IV contains the fault-finding capabilities of both test suites and Table III the execution times. Since we only have one value for each test suite, no analysis techniques are available and the tables in this section just present the measured data on which the answers to the research questions from Section III-A are based.

TABLE II. Descriptive measures for the FSM_x that is used to generate $\mathbf{TS}^x_{fittest}$

x	**Variable**	
A	Number of traces used to infer FSM	1
	Average trace length	13
	Number of states in generated FSM	7
	Number of different events, i.e. transition labels	6
	Number of transitions in the FSM	6
B	Number of traces used to infer FSM	1
	Average trace length	23
	Number of states in generated FSM	7
	Number of different events, i.e. transition labels	8
	Number of transitions in the FSM	8
C	Number of traces used to infer FSM	3
	Average trace length	263
	Number of states in generated FSM	64
	Number of different events, i.e. transition labels	31
	Number of transitions in the FSM	126

RQ1: *Compared to the current test suite used for testing at SOFTEAM, can the FITTEST technologies contribute to the* ***effectiveness*** *and* ***efficiency*** *of testing when it is used in the testing environments at SOFTEAM?*

Inspecting Table IV we can see that TS_{soft} finds only 3 more faults than $TS_{fittest}$ (2 with blocking severity and 1 with minor severity). On the other hand $TS_{fittest}$ finds one blocking fault that TS_{soft} does not find. This is a rather positive outcome for the FITTEST techniques, considering the fact that the design of the FITTEST test suite was mostly automated, only 1 hour for creating a DIS was needed (the collecting of the logs would normally be done during normal usage of the SUT and hence is not calculated within the testing time). One hour of automated test set up compared to 56 hours of manual test suite design is a huge difference in effort for the small difference in faulti-finding effectivity. Moreover, the evolvability of the logs implies that if these would evolve with more system usage, more faults could be found. This should be investigated over time.

RQ2: *How much effort would be required to deploy the FITTEST technologies within the testing processes implanted at SOFTEAM?*

SOFTEAM is positive about the effort to deploy the ITE, and concerning model inference, test generation and test

[5]http://www.berner-mattner.com/en/berner-mattner-home/products/cte

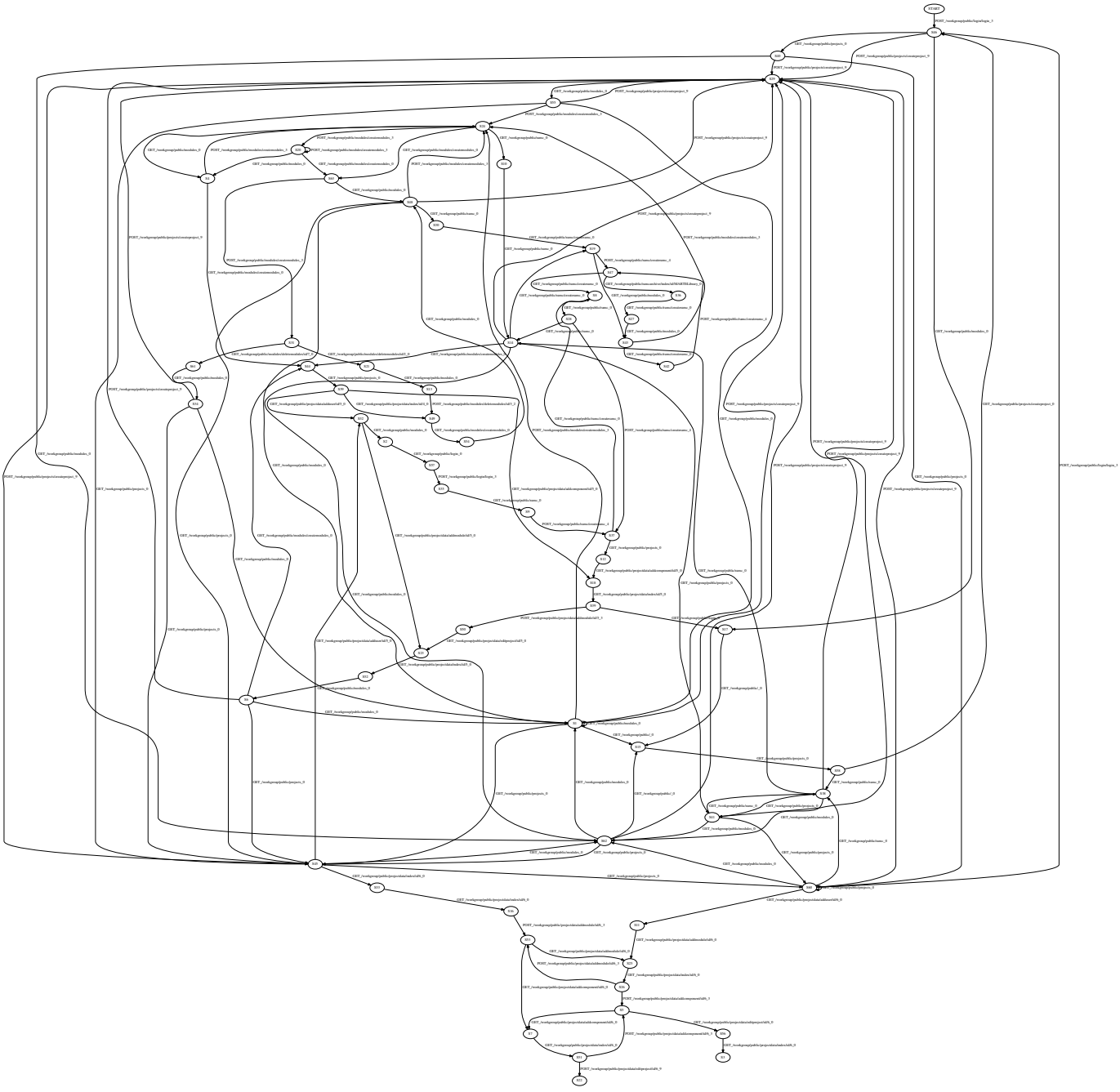

Fig. 4. A part of the model inferred for the case study. The model has 64 nodes and 126 transitions, inferred automatically. This figure illustrates how an inferred model looks like. In fact, it is machine generated and there is no need for a tester to read the model.

TABLE III. DESCRIPTIVE MEASURES RELATED TO SIZE AND EXECUTION TIME FOR THE TEST SUITES INVOLVED

	$\mathbf{TS}_{soft}$	$\mathbf{TS}^{A}_{fittest}$	$\mathbf{TS}^{B}_{fittest}$	$\mathbf{TS}^{C}_{fittest}$
number of abstract test cases	NA	1	3	65
number of concrete test cases	51	6	18	101
number of commands (or events)	294	30	84	685
execution Time with fault injection	230	2	6	35
execution Time without fault injection	60	2	6	35
code covered	86,63%	12,46%	17,17%	34,84%

execution which will definitely be deployed on SOFTEAM side they are confident that there is a real benefit to gain. However, a problem is foreseen with the logging activities, these should ideally take place at the client/user side but this is not in the hand of SOFTEAM. The laternative is collecting the logs at SOFTEAM as was done in this study.

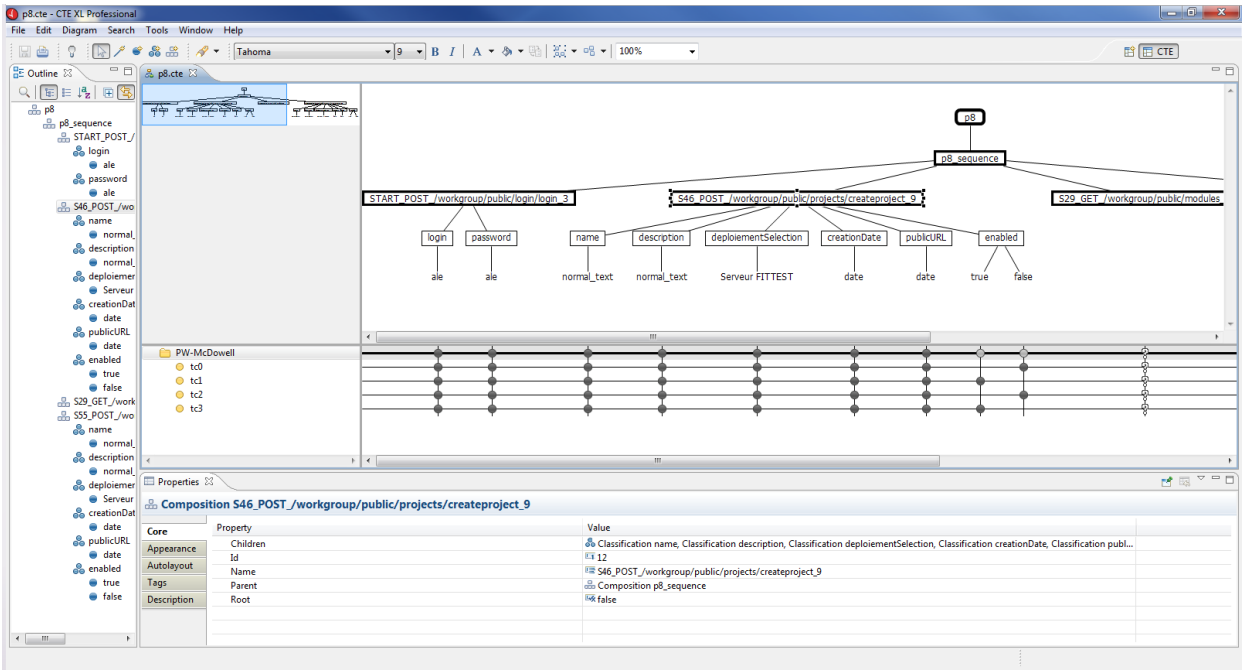

Fig. 5. An example of a test sequence and the test cases generated for the sequence. Each test sequence is transformed to a classification tree with input classifications. The combinations of input classes along the sequence are test cases that can be transformed to executable ones automatically.

TABLE IV. EFFECTIVENESS MEASURES FOR THE TEST SUITES WITH RESPECT TO THE INJECTED FAULTS. "0" MEANS THAT THE CORRESPONDING FAULT WAS NOT DETECTED, WHILE "1" MEANS IT HAS BEEN DETECTED.

TS	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	Total
$\mathbf{TS}_{soft}$	0	1	1	**1**	**1**	1	1	1	0	1	1	0	1	**1**	1	**1**	**1**	14
$\mathbf{TS}^{A}_{fittest}$	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
$\mathbf{TS}^{B}_{fittest}$	0	1	1	0	0	1	0	0	0	1	0	0	0	0	1	0	0	5
$\mathbf{TS}^{C}_{fittest}$	0	1	1	0	0	1	1	1	**1**	1	1	0	1	0	1	0	0	10

$\mathbf{TS}_{soft}$		$\mathbf{TS}^{A}_{fittest}$	$\mathbf{TS}^{B}_{fittest}$	$\mathbf{TS}^{C}_{fittest}$
generate the FSM	NA	automated $\leq$ 1sec of CPU	automated $\leq$ 1sec of CPU	automated $\leq$ 1sec of CPU
specify DIS	NA	20 mins	30 mins	60 mins
concrete tests	manual 56 hours	automated $\leq$ 1sec of CPU	automated $\leq$ 1sec of CPU	automated $\leq$ 1sec of CPU

TABLE V. DESCRIPTIVE MEASURES RELATED TO TIME FOR CONSTRUCTION FOR THE FITTEST TEST SUITES

RQ3: *How satisfied are SOFTEAM testing practitioners during the learning, installing, configuring and usage of the technique when it is used in their real testing environments?*

SOFTEAM indicated to be reasonably satisfied with the usage of the FITTEST technique in their real testing environments. However they indicated that this might be due to the fact that the practitioners are well familiar with the technologies behind Java/Eclipse so the new technology was very well accepted from the beginning of its introduction. From SOFTEAM point of view, the one thing that could be improved in the FITTEST prototype in order to take into real testing environment is the network configuration for the logging facilities.

B. Threats to validity

Internal validity. It is of concern when causal relations are examined. In our case study, an internal validity threat is related to the logs generated and used for automatically constructing the test models. Because of the quality of models can be affected by the content of the input logs. We are aware of this threat and have asked SOFTEAM for a diverse set of logs. Another similar threat is that the quality of concrete test cases can be affected by the completeness of the Domain Input Specification (DIS) file because incomplete specification will weaken the efficiency of the $\text{TS}_{fittest}$. In fact, this threat might affect the overall number of detected faults by $\text{TS}_{fittest}$, but if the specification can be improved, such number can be greater. Therefore, the conclusion about the effectiveness of the $\text{TS}_{fittest}$ remains unchanged. Regarding to the involved subjects from SOFTEAM, although they had a high level of expertise and experience working in the industry as testers, they had no previous knowledge of the FITTEST tools. This threat was reduced by means of a closer collaboration between FBK and SOFTEAM, by complementing their competences in order to avoid possible mistakes or misunderstandings.

External validity. It is concerned with to what extent it is possible to generalize the findings, and to what extent the findings are of interest to other people outside the investigated case. Our results rely on one industrial case study using a given set of artificial faults. Although running such studies is expensive in terms of time consuming, we plan to replicate it with in order to have a more generalizable conclusions. However, as discussed earlier, the system under testing used is a typical of a broad category of industrial systems with users of the management system.

Construct validity. This aspect of validity reflect to what extent the operational measures that are studied really represent what the researcher have in mind and what is investigated according to the research questions. This type of threat is mainly related to the use of injected faults to measure the fault-finding capability of our testing strategies. This is because the types of faults seeded may not be enough representative of real faults. In order to mitigate this threat, the SOFTEAM team identified representative faults that were based on real faults, identified in earlier time of the development. This identification although was realized by a senior tester, the list was revised by all SOFTEAM.

V. Conclusions

We have presented a "which is better" [8] case study for evaluating FITTEST testing tools with a real user and real tasks within a realistic industrial environment of SOFTEAM. The design of the case study has been done according to the methodological framework for defining case studies presented in [12]. Although this small scale case study will never provide general conclusions with statistical significance, the obtained results can be generalized to other similar software in similar testing environments of SOFTEAM [15], [7]. Moreover, the study was very useful for technology transfer purposes: some remarks during the study indicate that the FITTEST techniques would not have been evaluated in so much depth if it would not have been backed up by our case study design. Finally, having only limited number of subjects available, this study took several weeks to complete and hence we overcame the problem of getting too much information too late.

The objective of this research was to examine the advancements of the FITTEST tools and validate their potential to improve current testing practices at SOFTEAM. The following were the results of the case study:

- The FITTEST tools can increase the effectiveness of the current practice.
- The efficiency of the FITTEST tools is found acceptable by SOFTEAM.
- The test cases automatically generated by the FITTEST tools support better the identification of the source of the faults
- The effort for deploying the FITTEST within a real industry case has been found reasonable by SOFTEAM.

Moreover, from the FITTEST project's point of view we have the following results:

- The FITTEST tools have shown to be useful within the context of a real industrial case.
- The FITTEST tools have the ability to automate the testing process within a real industrial case.

Acknowledgment

This work was financed by the FITTEST project, ICT-2009.1.2 no 257574. Also, we would like to thank the great help we got from Alon Aradi for conducting the experiments.

References

[1] B. Beizer. *Software Testing Techniques*. International Thomson Computer Press, 1990.

[2] A. Biermann and J. Feldman. On the synthesis of finite-state machines from samples of their behavior. *IEEE Trans. on Computers*, 21(6), 1972.

[3] A. C. Dias Neto, R. Subramanyan, M. Vieira, and G. H. Travassos. A survey on model-based testing approaches: a systematic review. In *Proceedings of the 1st ACM international workshop on Empirical assessment of software engineering languages and technologies: held in conjunction with the 22nd IEEE/ACM International Conference on Automated Software Engineering (ASE) 2007*, WEASELTech '07, pages 31–36, New York, NY, USA, 2007. ACM.

[4] D. Graham and M. Fewster. *Experiences of Test Automation*. Pearson, 2012.

[5] M. Grochtmann and J. Wegener. Test case design using classification trees and the classification-tree editor cte. In *Proceedings of the 8th International Software Quality Week*, San Francisco, USA, Mai 1995.

[6] M. Harman and B. F. Jones. Search-based software engineering. *Information and Software Technology*, 43(14):833–839, 2001.

[7] W. Harrison. Editorial (N=1: an alternative for software engineering research). *Empirical Software Engineering*, 2(1):7–10, 1997.

[8] B. Kitchenham, L. Pickard, and S. Pfleeger. Case studies for method and tool evaluation. *Software, IEEE*, 12(4):52 –62, July 1995.

[9] G. J. Myers. *The Art of Software Testing*. John Wiley and Sons, 1979.

[10] C. D. Nguyen, A. Marchetto, and P. Tonella. Combining model-based and combinatorial testing for effective test case generation. In *Proceedings of the 2012 International Symposium on Software Testing and Analysis*, pages 100–110. ACM, 2012.

[11] C. Nie and H. Leung. A survey of combinatorial testing. *ACM Comput. Surv.*, 43:11:1–11:29, February 2011.

[12] T. Vos, B. Marín, I. Panach, A. Baars, C. Ayala, and X. Franch. Evaluating software testing techniques and tools. In *Actas de XVI JISBD*, pages 531–536, 2011.

[13] T. E. J. Vos, B. Marín, M. J. Escalona, and A. Marchetto. A methodological framework for evaluating software testing techniques and tools. In *12th International Conference on Quality Software, Xi'an, China, August 27-29*, pages 230–239, 2012.

[14] T. E. J. Vos, P. Tonella, J. Wegener, M. Harman, W. Prasetya, and S. Ur. Testing of future internet applications running in the cloud. In S. Tilley and T. Parveen, editors, *Software Testing in the Cloud: Perspectives on an Emerging Discipline*, pages 305–321. 2013.

[15] A. Zendler, E. Horn, H. SchwŁrtzel, and E. Pldereder. Demonstrating the usage of single-case designs in experimental software engineering. *Information and Software Technology*, 43(12):681 – 691, 2001.